Two New Heuristics for Two-Sided Bipartite Graph Drawing*

Matthew Newton[1], Ondrej Sýkora[1], and Imrich Vrťo[2]

[1] Department of Computer Science, Loughborough University
Loughborough, Leicestershire LE11 3TU, The United Kingdom
{m.c.newton,o.sykora}@lboro.ac.uk, Fax: +44 (0)1509 211 586
http://parc.lboro.ac.uk/
[2] Institute of Mathematics, Slovak Academy of Sciences
Dúbravská 9, 842 35 Bratislava, Slovak Republic
vrto@savba.sk, Fax: +421 2 59306522

Abstract. Two new heuristic strategies are studied based on heuristics for the linear arrangement problem and a stochastic hill-climbing method for the two-sided bipartite crossing number problem. These are compared to the standard heuristic for two-sided bipartite drawing based on iteration of the barycentre method. Our experiments show that they can efficiently find good solutions.

1 Introduction

Graph drawing addresses the problem of finding a layout of a graph that satisfies given aesthetic and readability objectives. One standard problem with the drawing of bipartite graphs is that of two layer automatic drawing where the two vertex partitions are put in distinct points on two parallel lines and edges are drawn as straight line segments between the two lines. This type of drawing is the basic building block used for drawing hierarchical graphs [5,14,20] or producing row-based VLSI layouts [17]. The most important objective is probably minimisation of the number of crossings in the drawing, as the aesthetics and readability of graph drawings depend on the number of edge crossings (see [16]). VLSI layouts containing less crossings are more easily realisable and consequently cheaper. There are two basic variants of the problem: one-sided and two-sided crossing minimisation. In the one-sided problem the vertices of one part of the bipartite graph are placed in fixed positions on one line. The vertices of the other part are placed on the other line and their positions are found so that the number of pairwise edge crossings is minimised. The two-sided bipartite crossing number problem (or just the bipartite crossing number problem) is if the vertices on both sides are not restricted. Unfortunately, both problems are NP-hard [7,8]. A lot of different methods have been designed to solve the one-sided problem—heuristics

* Research of all the authors was supported by the EPSRC grant GR/R37395/01. Research of the last two authors was supported by the Slovak Scientific Grant Agency grant No. 2/7007/20.

M.T. Goodrich and S.G. Kobourov (Eds.): GD 2002, LNCS 2528, pp. 312–319, 2002.

and approximation algorithms (see e.g. [1,4]). For the two-sided bipartite problem a common technique is to apply a one-sided method iteratively. In [10] the one-sided methods, barycentre, LR-Opt (branch and cut method of [10]), split, median, stochastic, greedy-switch, assign and greedy-insert were applied iteratively until there was no change, i.e. until a local minimum had been achieved. The resulting ranking of the heuristics was as above. The iterated barycentre method was the best among the tested methods both in terms of quality and computation time. An interesting observation by the authors of [10], for graphs of up to about 15 vertices on both sides, was that for the graphs with a density of 10%, the resulting number of crossings were 5 to 33 times larger than the optimal value.

In this paper we suggest two new heuristic strategies. The first is based on the result in [18] showing a strong relation between the linear arrangement problem of a bipartite graph and its bipartite crossing number. We used two different methods to find a solution to the linear arrangement problem which implies respective permutations of vertices on both sides (layers). One (linear arrangement based on Fiedler's vector: LAF-method) is based on computing the graph's second laplacian eigenvalue and its corresponding vector, usually called Fiedler's vector [9]. The other (linear arrangement based on Koren and Harel's method: LAKH-method) is based on the multiscale method of Koren and Harel [12]. The second strategy, stochastic hill-climbing (SH)-method, is based on hill-climbing, a standard optimisation technique, and randomised swapping of vertices. In comparison with genetic algorithms or simulated annealing, stochastic hill-climbing is simpler and faster. Our method is different from the method of [3], which is also called 'stochastic' in [10]. It also differs from the greedy-switch heuristic of [6] and from sifting [13] which are based on the swapping of neighbouring vertices.

In our experiments we compared the above three methods with the iterated barycentre (IB)-method. The results show that both LA*-methods usually outperform the IB-method, especially LAKH-method, which produces good drawings in an acceptable running time. The SH-method often produces the best results (see Table 1, shown instead of a graphical comparison due to lack of space).

2 Notations

Let $G = (V, E)$, $V = V_0 \cup V_1$ be a bipartite graph with vertex partitions V_0 and V_1. A *bipartite drawing* of G is obtained by placing the vertices of V_0 and V_1 into distinct points on two horizontal lines y_0 and y_1, respectively, and drawing each edge with one straight line segment. Any bipartite drawing of G is identified by two permutations π_0 and π_1 of the vertices on y_0 and y_1. The problem of the two-sided bipartite drawing of G is to find permutations π_0 and π_1 that minimise the number of pairwise edge crossings in the corresponding bipartite drawing. The problem of the one-sided bipartite drawing of G is the same, except the permutation π_1 is fixed.

Let $\mathrm{bcr}(G, \pi_0, \pi_1)$ denote the total number of crossings in the bipartite drawing represented by the permutations π_0 and π_1. The *bipartite crossing number* of G, denoted by $\mathrm{bcr}(G)$, is the minimum number of crossings over all π_0 and π_1. Clearly, $\mathrm{bcr}(G) = \min_{\pi_0,\pi_1} \mathrm{bcr}(G, \pi_0, \pi_1)$.

Given an arbitrary graph $G = (V, E)$, the linear arrangement problem is to determine a bijection $f : V \to \{1, 2, 3, ..., |V|\}$ such that $\sum_{uv \in E} |f(u) - f(v)|$ is minimised. This minimum value, whose computation is NP-hard [8], is denoted by $\mathrm{L}(G)$.

Furthermore, d_v denotes the degree of v, δ_G the minimum degree of G and Δ_G, the maximum degree of G.

3 Linear Arrangement Problem Based Methods

The bipartite crossing number problem was studied in [18] and a connection between this problem and the linear arrangement problem was established. Lower and upper bounds for bipartite crossing number $\mathrm{bcr}(G)$ were derived, where the main term is the optimal arrangement value:

Theorem 1 *Let $G = (V_0, V_1, E)$, then*

$$\frac{1}{36}\delta_G \mathrm{L}(G) - \frac{1}{12}\sum_{v \in V} d_v^2 \leq \mathrm{bcr}(G) \leq 5\Delta_G \mathrm{L}(G).$$

By using this result, a polynomial time approximation algorithm [18] with performance guarantee $\mathrm{O}(\log n)$ from the optimal was obtained for the bipartite crossing number problem if $\Delta_G = O(\delta_G)$. The algorithm is based on an approximation algorithm for the linear arrangement problem, where an approximated solution of the linear arrangement problem is found and then the vertices of the partitions of V_0 and V_1 are put on the lines y_0 and y_1 in the same order as produced for the linear arrangement problem. Using this idea, we can use any heuristic that solves the linear arrangement problem to produce the pair of permutations π_0 and π_1.

In [9] a heuristic is suggested that solves the linear arrangement problem by computing the corresponding (so called Fiedler's) eigenvector f to the smallest positive Laplacian eigenvalue λ_G of the graph G. Recall that the Laplacian of a graph G is the matrix $I(G) - A(G)$, where $A(G)$ is the adjacency matrix of the graph G and $I(G)$ is the diagonal matrix with vertex degrees on the diagonal, i.e. $i_{vv} = d_v$, and $i_{uv} = 0$ if $u \neq v$ and the Fiedler's eigenvector f fulfills: $(I(G) - A(G))f = \lambda_G f$. Using this approximation of linear arrangement we get the LAF-method shown in Algorithm 1.

The other method (LAKH-method) is based on the multiscale method of Koren and Harel [12] that computes an approximation of the linear arrangement.

4 Stochastic Hill-Climbing Method

The space of orderings of vertices of both sides for two-sided bipartite drawing problem can be represented by all pairs of permutations of $\{0, 1, 2, \ldots, n_0 - 1\}$,

Algorithm 1. "Linear Arrangement based on Fiedler's vector" (LAF-method)

```
find smallest positive Laplacian eigenvalue λ_G or a good approximation
find corresponding eigenvector (Fiedler vector)
construct approximation to the solution of the Linear Arrangement Problem
construct permutations π_0 and π_1
evaluate the number of crossings for the pair of permutations
```

and of $\{0, 1, 2, \ldots, n_1 - 1\}$, where n_0 and n_1 are the number of vertices in V_0 and V_1, respectively. To find a solution of the two-sided bipartite drawing problem means searching this space for a pair of permutations providing the minimum number of crossings. To move across the space we might use some type of steps, evaluate a new visited pair of permutations, accept it if it fulfills a condition (e.g. it has lower or equal number of crossings) and continue to search from the new accepted permutation. If the new permutation was not accepted we continue from the old one. The starting permutations are given by the input. Algorithm 2 shows the specific method we used.

Algorithm 2. "Stochastic Hill-Climbing Method" (SH-method)

```
repeat until (termination-condition)
  randomly choose one of the sides 0 or 1
  randomly choose two different vertices on the chosen side
  swap their positions
  evaluate the number of crossings for the new pair of permutations
  if the  number of crossings decreases
  then
    keep the new pair of permutations
  else
    return to the previous pair of permutations
```

The termination condition can be defined by the number of iterations, or the number of iterations that have passed since the last swap that gave an improvement to the number of crossings (the number of stagnations). According to our experience, the termination should happen if the number of stagnations exceeds a value $\alpha * density/(density + \beta)$ where $density = |E|/|V|$, and α, β are proper constants. For the sizes of our graphs these constants could be: $\alpha = 1350, \beta = 65$.

In our experiments we ran the SH-method while the number of crossings was higher than that achieved by the IB-method, the LAF-method and LAHK-method. The SH-method program also had a maximum time limit of five minutes and a few graphs did not complete within this limit. For some graphs SH-method stagnated and did not reach the number of crossings of the IB (this includes e.g. `leonardo-90x64` and `hyper-9`) or LA$*$-methods (this includes e.g. `bcspwr01` and

`cycle-100`). An asterisk in Table 1 indicates one of the previous two conditions. Some graphs completed in less than 0.00005 second, so due to rounding it looks like they finished immediately (`climb-100x100-t10` and `ball-100x100`). The number of crossings that each algorithm reached, together with the time taken, is shown in Table 1; IB-method in columns two and three, LAF-method in columns five and six, LAKH-method in columns eight and nine. The running times of SH-method necessary to achieve the number of crossings equal to or below that of the IB–, LAF– and LAKH-methods are shown in columns four, seven and ten, respectively. The best number of crossings found for each graph is shown in boldface.

5 Experiments

Our algorithms were implemented in C, running on 50 Sun ULTRAsparc 5 workstations running Solaris 7. The algorithms were compiled with the GNU C compiler version 2.95.2. One of the computers ran a job control system, written in Perl, to run experiments continuously on the other machines.

The experiments for the IB–, LAF– and LAKH-methods were run first. These programs output the pair of permutations of the nodes in each graph, π_0 followed by π_1, from which the number of crossings was counted. Before each experiment, π_0 and π_1 were randomised (this did not affect the LA∗-methods, which do not need an initial permutation).

The quality of the results of the LA∗-methods depends on the quality of the approximation of $\mathrm{L}(G)$ for a graph G. To compute the Fiedler's eigenvector in the LAF-method we used the procedure `jacobi` from Numerical Recipes [15]. Although the `jacobi` procedure is not too efficient, it gives trustworthy results and is recommended for problems of small-to-moderate order. Typical $n \times n$ matrices require $12n^3$ to $20n^3$ operations. In the LAKH-method we used the multiscale algorithm of Koren and Harel [12] for solving the linear arrangement problem running in linear time in the graph size ($|V| + |E|$). Apart from meshes, the LAKH-method gave better results than the LAF-method. It is also much faster, especially for larger graphs. Thanks are due to Yehuda Koren who kindly provided us their procedure so we were able to test this method.

All IB–, LAF– and LAKH-methods were run ten times on each graph and the average number of crossings and running time were recorded.

A new set of experiments were then run for the SH-method. The algorithm was run thirty times for each graph. For the first, second and third ten runs the program halted when the number of crossings was equal to or below that of the IB–, LAF–, LAKH-methods, respectively. As in the first set of experiments, π_0 and π_1 of each graph were randomised before each test run.

Graph Test Sets. For our experiments we used the classes of graphs from [4] that contain 'synthetic' bipartite graphs with a known bipartite crossing number, bipartite graphs extracted from digitised images (originally suggested by Knuth in the Stanford Graphbase [11]) and from test data retrieved from US National Institute of Standards and Technology [2]. For our tests we also added

the following graphs for which we either know, or believe we know, the optimum drawings. Among these are cycles (with bipartite crossing number $\frac{n}{2}-1$, if there are n vertices in the cycle, n is even), rectangular meshes, square $n \times n$ meshes (with supposed bipartite crossing number $\frac{4}{3}n(n-1)(n-2)+1$) and hypercubes (see [19]). Finally, we also used randomly generated graphs.

Random Graphs. The random graphs we used were generated based on edge density. The generator takes two arguments: n, the number of nodes the graph should have, and $p \in [0..100]$ that determines the percentage chance that an edge will exist between two nodes. n must be an even number, and the graph is generated with $\frac{n}{2}$ nodes in each part. The general graphs were generated having sizes $n \in [20, 40, 60, 80, 100, 200, 400, 600, 800, 1000]$ and densities of 10%, 1% and 0.1%.

Acknowledgement

The authors wish to thank Yehuda Koren and Irene Finocchi, for making available their software, and Mark Withall for useful discussions during the preparation of this paper.

Table 1. Comparisons between IB-method (columns two and three), LAF-method (columns five and six), LAKH-method (columns eight and nine) and SH-method. In columns four, seven and ten are the running times necessary for SH-method to achieve the number of crossings equal to or below that of the IB–, LAF– and LAKH-methods, respectively. Time is in seconds.

Graph Name	IB-method		SH to IB	LAF-method		SH to LAF	LAKH-method		SH to LAKH
	Crossings	Time	Time	Crossings	Time	Time	Crossings	Time	Time
a-100x100	2550802	0.0031	0.3989	2716464	19.3547	0.0111	**1301275**	15.888	*
a-87x81	2414484	0.0025	0.8535	1327488	2.5816	25.6836	**1301275**	15.034	*
add20	3621791	0.4223	0.0088	1177229	1296.39	51.7084	**1021504**	37.708	*
andywarhol-100x65	**1893498**	0.0074	*	2856377	2.248	11.4422	2784605	17.056	35.1931
ball-100x100	947575	0.004	0	934566	9.413	0.0068	**749982**	9.038	22.3192
barpermute-100x100	3175608	0.004	2.6106	4161703	5.4557	0.0162	**2040676**	18.829	61.6639
bars-100x100	3517232	0.0325	0.0324	**2075771**	6.7551	*	2153777	21.607	45.9696
bcspwr01	2954	0.0019	0.0029	464	0.2698	*	**422**	1.263	*
bcsstk15	889155	0.1933	*	**727592**	923.3729	*	741208	36.756	*
brick-100x100	501133	0.0171	0	473653	24.6587	0.0011	**248575**	10.123	3.3267
camil-100x78	**1559231**	0.0078	*	2076490	3.7254	1.2589	1708444	15.594	12.8803
can61	69842	0.0026	0.0075	**12788**	1.1197	*	12847	3.835	*
cbar-100x100	**88815**	0.0027	*	**88815**	6.4762	*	**88815**	8.597	91.8295
cbar-20x20	465	0.0004	0.0018	**19**	0.0314	*	**19**	0.515	*
cbar-50x50	**3870**	0.0015	19.2498	**3870**	0.6231	21.2854	**3870**	2.64	26.1418
climb-100x100-t10	19276	0.0077	0	10824	10.7103	0.0212	**4764**	3.768	*
climb-100x100-t100	544406	0.0066	0.0182	536316	29.5412	0.0359	**252671**	9.847	*
climb-100x100-t110	672990	0.0071	0	604213	6.1009	0.0876	**307040**	10.205	10.0924
climb-100x100-t120	760146	0.0062	0.0116	655080	5.3363	0.243	**367847**	11.468	7.8663
climb-100x100-t130	989397	0.0093	0	923308	6.1088	0.0194	**493285**	12.783	6.3544
climb-100x100-t140	1355640	0.0138	0	1081502	5.6321	0.4582	**694349**	14.181	11.065
climb-100x100-t150	2627049	0.0155	0	1690572	5.7368	11.4668	**1569210**	17.33	20.2285
climb-100x100-t160	3796488	0.0297	0	2697887	5.6074	8.7222	**2597067**	19.739	20.7422
climb-100x100-t20	83505	0.0087	0	39548	11.8175	1.0374	**35877**	5.512	0.4004
climb-100x100-t30	136135	0.0062	0.0022	75347	14.7937	0.8598	**70994**	6.53	0.8385
climb-100x100-t40	169887	0.0082	0.0087	100168	15.6475	0.7436	**96179**	6.581	0.5109
climb-100x100-t50	217630	0.0078	0.038	203147	18.0365	0.0756	**124927**	7.506	0.9283
climb-100x100-t60	239888	0.0067	0.0406	151225	19.583	0.782	**128702**	7.294	2.0096
climb-100x100-t70	271755	0.0066	0.05	178569	20.3396	1.0076	**156352**	7.927	1.025
climb-100x100-t80	268910	0.0076	0.1685	298802	21.2215	0.0607	**174073**	8.199	1.1534
climb-100x100-t90	341906	0.0084	0.0457	224044	24.6957	0.7863	**184132**	8.31	2.4242
climb-100x100	997171	0.0084	0	914106	5.539	0.0136	**471565**	12.415	13.5789
climb-20x20	1316	0.0004	0	1084	0.0322	0.0004	**536**	0.659	0.0221
climb-40x40	24032	0.0016	0.0001	19797	0.3074	0.0076	**11464**	2.31	0.1219
climb-60x60	131937	0.004	0	109592	0.9228	0.0246	**64596**	6.069	1.0191
climb-80x80	415731	0.005	0	335135	2.2201	0.1762	**211614**	8.251	2.6622
cycle-100	467	0.0043	*	**49**	0.7583	*	83	1.53	*
cycle-1000	6639	1.1338	*	**499**	4875.714	*	778	17.825	*
cycle-20	48	0.0001	0.0004	**9**	0.009	*	13	0.168	*
cycle-200	1057	0.0198	*	**99**	10.3766	*	171	3.355	*
cycle-300	1813	0.054	*	**149**	41.6743	*	250	5.077	*

Graph Name	IB-method		SH to IB	LAF-method		SH to LAF	LAKH-method		SH to LAKH
	Crossings	Time	Time	Crossings	Time	Time	Crossings	Time	Time
cycle-40	122	0.0005	0.0053	**19**	0.0457	*	29	0.474	*
cycle-400	2228	0.0981	*	**199**	152.9575	*	328	7.116	*
cycle-500	2987	0.2015	*	**249**	282.9466	*	404	8.998	*
cycle-60	274	0.0014	0.0104	**29**	0.1782	*	45	0.8	*
cycle-80	253	0.0024	0.0684	**39**	0.4786	*	63	1.185	*
geom2-100x100	179880	0.0037	0.1256	391832	2.4829	0	**43073**	6.538	*
hyper-10	**2221867**	0.2571	*	2548526	7116.157	144.4579	*	*	*
hyper-4	131	0.0001	0.0011	**118**	0.0049	0.0022	124	0.154	0.0011
hyper-5	999	0.0002	0.0018	**776**	0.0346	0.0143	786	0.545	0.0093
hyper-6	4710	0.0006	0.0395	**4200**	0.1669	*	4359	1.494	0.0845
hyper-7	22714	0.0026	*	23464	16.9141	0.526	**22134**	3.957	1.2749
hyper-8	108038	0.013	*	**106400**	38.4887	*	106922	9.756	9.6853
hyper-9	**494357**	0.0541	*	535292	1247.529	24.5621	498715	21.38	70.8947
irregular-100x100	886963	0.0027	6.6072	3690604	5.2365	0	**492835**	12.699	*
ladyshoe-67x100	883543	0.0054	0.9114	727351	2.0314	2.7296	**669064**	11.151	13.8607
leonardo-90x64	**455396**	0.0077	*	1341314	1.8802	0.3251	1073758	11.988	4.9457
mahindas	1318184	0.8055	0.0013	1266715	836.6369	299.3661	**324736**	25.755	*
mesh-10-10	1212	0.0028	*	**997**	0.7666	*	1009	1.978	*
mesh-15-15	6286	0.0175	*	3897	23.6179	*	**3858**	5.614	*
mesh-20-20	12570	0.0701	*	**9519**	118.2442	*	9993	8.964	*
mesh-25-25	30245	0.2926	*	**19340**	551.8628	*	19708	14.722	*
mesh-30-10	6246	0.0445	*	3729	31.9283	*	**3367**	6.287	*
mesh-30-15	16014	0.1238	*	8866	182.8301	*	**8119**	10.96	*
mesh-30-20	27610	0.2655	*	16465	650.1603	*	**14920**	13.7	*
mesh-30-25	39139	0.5404	*	25258	1056.526	*	**23524**	18.843	*
mesh-30-30	45950	0.654	*	**34200**	2470.742	*	34836	23.848	*
mesh-30-5	1225	0.0066	*	706	2.3688	*	**679**	2.946	*
mesh-35-35	85203	1.5665	*	**55098**	6805.752	*	*	*	*
mesh-40-40	120812	3.2542	*	**84679**	21971.12	*	*	*	*
mesh-5-5	252	0.0002	0.0007	**84**	0.0088	*	90	0.302	0.0124
monnalisa-100x65	**1945030**	0.0101	*	5179482	2.7837	0.4914	4314354	19.776	93.816
monnalisa-20x13	**3588**	0.0003	*	6332	0.021	0.0336	6156	0.919	*
monnalisa-40x26	**50578**	0.0023	*	128954	0.153	0.0044	106336	3.903	*
monnalisa-60x39	**257956**	0.0035	*	656433	0.5054	0.0472	540595	7.249	11.2691
monnalisa-80x52	**770234**	0.0077	*	2072231	1.3043	0.2279	1731778	12.19	56.0786
moonsurface-100x100	2146803	0.0175	0.7537	1807232	5.771	3.1508	**1599527**	16.847	36.2476
orani678	4941969	0.9359	0.0601	2202672	710.6326	26.2889	**1955821**	36.533	88.4403
psmigr1	2.05e+09	0.265	73.6012	1.91e+09	608.8471	*	**1.85e+09**	423.455	*
random-100-0.1	**0**	0.0005	0	**0**	0.1306	0	**0**	0	0
random-100-1	102	0.0009	0.0001	102	0.0126	0	**0**	1.201	*
random-100-10	16707	0.0074	0.0003	9575	0.7139	0.0667	**8304**	2.706	1.2585
random-1000-0.1	19093	0.3769	0.0049	18980	3.6272	0.0039	**0**	0	*
random-1000-1	1683123	9.4952	0.0113	1672515	3937.91	0.009	**772973**	41.513	*
random-1000-10	1.58e+08	6.0268	1.243	1.4e+08	3973.301	197.7546	**1.36e+08**	226.149	*
random-20-0.1	**0**	0.0001	0	**0**	0.0001	0	**0**	0	0
random-20-1	**0**	0.0001	0	**0**	0.0001	0	**0**	0.075	0
random-20-10	25	0.0001	0	12	0.0007	0.0003	**0**	0.126	*
random-200-0.1	5	0.0016	0.0006	5	0.4618	0.0011	**0**	0	0.0092
random-200-1	2257	0.0119	0	1748	0.1166	0.0017	**0**	0	*
random-200-10	275147	0.0446	0.0011	201022	6.843	0.4308	**182662**	9.832	3.8573
random-40-0.1	1	0.0001	0	1	0.0003	0	**0**	0.286	0.0001
random-40-1	3	0.0001	0	1	0.0106	0.0003	**0**	0.375	0.0004
random-40-10	370	0.0004	0.0002	268	0.0217	0.0006	**87**	0.498	*
random-400-0.1	400	0.0154	0.0006	288	0.6268	0.0032	**0**	0	*
random-400-1	42140	0.442	0.0006	21995	234.6991	0.0763	**5032**	7.745	*
random-400-10	4210581	0.6203	0.0424	3435193	118.4158	5.4909	**3215659**	35.652	41.3217
random-60-0.1	**0**	0.0002	0	**0**	0.0261	0	**0**	0	0
random-60-1	13	0.0002	0.0001	8	0.0263	0.0004	**0**	0.594	0.0021
random-60-10	1798	0.0015	0.0002	2018	0.164	0	**376**	1.011	*
random-600-0.1	2806	0.1033	0.0013	2497	0.92	0.0035	**0**	0	*
random-600-1	204855	0.8702	0.0043	206670	611.0068	0.001	**58159**	15.637	*
random-600-10	20385097	2.5476	0.0854	17455934	715.8014	299.6423	**16400000**	81.408	*
random-80-0.1	**0**	0.0003	0	**0**	0.0721	0	**0**	0	0
random-80-1	31	0.0005	0.0002	15	0.0151	0.0011	**0**	0.9	*
random-80-10	5416	0.0041	0.0002	2143	0.3074	*	**2011**	1.673	*
random-800-0.1	6687	0.1985	0.0032	6851	2.5984	0.0036	**0**	0	*
random-800-1	620494	6.4117	0.0066	555193	2080.685	0.0794	**233566**	25.781	*
random-800-10	64476577	3.8251	0.1718	56162991	2123.121	299.6723	**53900000**	140.667	*
ring-100x100	1376092	0.0053	0.0032	1389508	4.7603	0	**344728**	11.287	*
ring-80x80	12979	0.0025	0.0004	10506	0.1934	0.0035	**338**	2.731	*
scra-100x100	2754221	0.0034	0	2558436	3.6129	0.3933	**1301275**	16.005	*
scra-87x81	2322384	0.0023	1.0449	1327488	2.4466	28.4834	**1301275**	15.098	*
scrball-100x100	928260	0.0034	0.0315	934566	9.1585	0.0048	**749982**	9.185	23.8414
texture1-100x100	3894302	0.0612	0.0167	3323967	6.5414	5.6343	**3211303**	21.713	21.8718
texture2-100x100	814985	0.0215	0.0005	432977	5.7274	2.2476	**390889**	11.83	13.0504
texture3-100x100	5035435	0.0327	0.0182	4238284	6.0099	7.0197	**4024066**	23.237	32.7983
texture4-100x100	2226143	0.0112	0	**1073482**	5.5928	28.7785	1080003	15.952	40.5893
texture5-100x100	2224888	0.0284	0.0003	1797332	6.7338	3.0323	**1636037**	17.318	36.0044
texture6-100x100	5572581	0.01	0.088	3745269	5.788	31.4037	**3744003**	22.265	60.4353
texture7-100x100	2900151	0.018	0	2319081	5.1494	3.2826	**2075480**	17.833	18.7751
tie-100x100	3190315	0.0059	0.2814	3184326	5.0847	0.3251	**2033573**	16.586	*
triangle-100x100	6123591	0.0029	0.008	**4003275**	6.5481	*	**4003275**	20.285	*
zenios	6845283	0.6242	1.0495	1113354	1220.934	*	**987540**	33.587	*

References

1. Bastert, O., Matuszewski, C.: Layered drawings of digraphs. In: Drawing Graphs, Methods and Models (eds. Kaufmann, M., Wagner D.), LNCS 2025, Springer (2001) 87–118
2. Boisvert, R., Pozo, R., Remington, K., Barrett, R., Dongarra, J.: Matrix Market: a web resource for test matrix collections. In: The Quality of Numerical Software: Assessment and Enhancement (ed. Boisvert, R.), Chapman and Hall, London, (1997) 125–137, Matrix Market is available at the URL: `http://math.nist.gov/MatrixMarket/`
3. Dresbach, S.: A new heuristic layout algorithm for DAGs. In: Operations Research Proceedings 1994 (eds. Derigs, U., Drexl, A.B.A.), Springer (1994) 121–126
4. Demetrescu, C., Finocchi, I.: Removing cycles for minimizing crossings. J. Experimental Algorithmics, to appear
5. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for Visualization of Graphs. Prentice Hall (1999)
6. Eades, P., Kelly, D.: Heuristics for reducing crossings in 2-layered networks. Ars Combin. **21** (1986) 89–98
7. Eades, P., Wormald, N.: Edge crossings in drawings of bipartite graphs. Algorithmica **11** (1994) 379–403
8. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM J. Algebraic Disc. Meth. **4** (1983) 312–316
9. Juvan, M., Mohar, B.: Optimal linear labellings and eigenvalues of graphs. Discrete Appl. Math. **36** (1992) 153–168
10. Jünger, M., Mutzel, P.: 2-Layer straight line crossing minimization: Performance of exact and heuristic algorithms. J. Graph Algorithms Appl., **1** (1997) 1–25
11. Knuth, D.E.: The Stanford GraphBase: A platform for combinatorial computing. Addision-Wesley, (1993)
12. Koren, Y., Harel, D.: A multi-scale algorithm for the linear arrangement problem. In: 28th Intl. Workshop on Graph-Theoretic Concepts in Computer Science (WG'2002), LNCS, Springer (2002), to appear
13. Matuszewski, C., Schönfeld, R., Molitor, P.: Using sifting for k-layer straightline crossing minimization. In: 7th Intl. Symp. on Graph Drawing (GD'99), LNCS 1731, Springer (1999) 217–224
14. Mutzel, P.: Optimization in leveled graphs. In: Pardalos, M., Floudas C.A. (eds.): Encyclopedia of Optimization. Kluwer, Dordrecht (2001)
15. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C. The Art of Scientific Computing. Second edition, Cambridge University Press (1992) `http://www.nr.com/`
16. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: 5th Intl. Symp. on Graph Drawing (GD'97), LNCS 1353, Springer (1998) 248–261
17. Sarrafzadeh, M., Wong, C.K.: An Introduction to VLSI Physical Design. McGraw Hill, (1996)
18. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrťo, I.: On bipartite drawings and the linear arrangement problem. SIAM J. Computing **30** (2001), 1773–1789
19. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrťo, I.: A new lower bound for the bipartite crossing number with algorithmic applications. Theoretical Computer Science **245** (2000) 281–294
20. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics **SMC-11** (1981) 109–125